SPRAWOZDANIE

Zajęcia: Nauka o danych II

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium Nr 2 Data 29.03.2025 Temat: „Praktyczne ćwiczenia z algorytmami optymalizacji dla uczenia maszynowego” Wariant 2	Anna Więzik Informatyka II stopień, niestacjonarne, 2 semestr, gr.1a TTO

1. Polecenie:

Link do repozytorium: https://github.com/AnaShiro/NoD2_2025

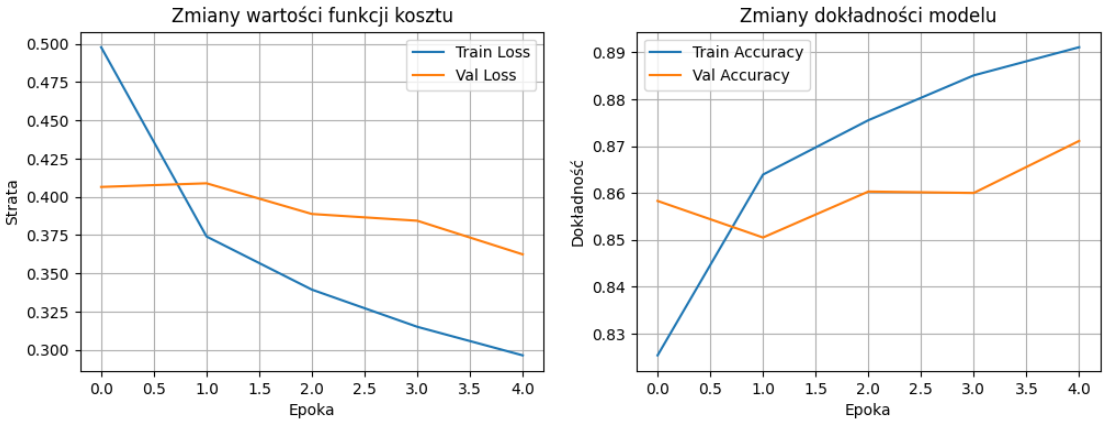
2. **Wariant 2**
   - Porównaj uczenie z $\eta = 0.05, 0.01, 0.005$.
   - Użyj funkcji: $f(x, y) = \sin(x)\cos(y) + x^2$.
   - Przetestuj TensorBoard dla modelu MLP klasyfikującego zbiór Fashion-MNIST.

Optymalizacja w uczeniu maszynowym sprowadza się do minimalizacji funkcji kosztu L(θ), gdzie θ oznacza parametry modelu. Kluczową rolę odgrywa gradient ∇L, który wskazuje kierunek najszybszego spadku wartości funkcji. Klasyczny Gradient Descent aktualizuje θ po każdej pełnej ocenie zbioru danych, co zapewnia stabilność kosztem wolniejszej konwergencji. Stochastic Gradient Descent przyspiesza proces, szacując gradient na losowych przykładach lub małych partiach, wprowadzając jednak większą wariancję aktualizacji. Momentum dodaje „bezwładność”, wygładzając ścieżkę spadku i redukując oscylacje. RMSProp adaptuje krok uczenia poprzez normalizację gradientów względem ich bieżącej wariancji, a Adam łączy zalety Momentum i RMSProp, dodatkowo korygując estymaty momentów, dzięki czemu zwykle wymaga minimalnego ręcznego strojenia hiperparametrów. Dobór algorytmu zależy od charakterystyki danych, pożądanej szybkości zbieżności i tolerancji na niestabilność przebiegu uczenia.

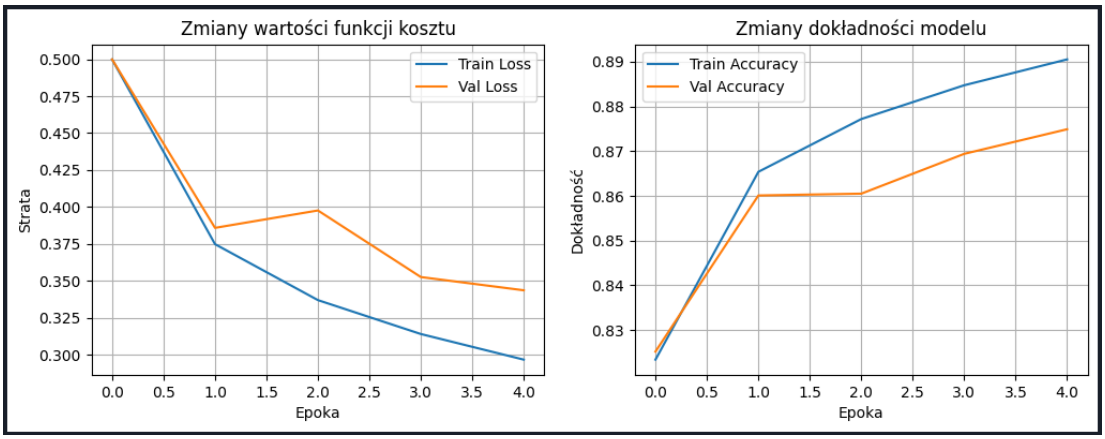
2. Opis programu opracowanego

```python
import tensorflow as tf
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
x_train, x_test = x_train.reshape(-1, 28*28) / 255.0, x_test.reshape(-1, 28*28) / 255.0

model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation="relu", input_shape=(784,)),
    tf.keras.layers.Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])

history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test), verbose=0)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Val Loss")
plt.xlabel("Epoka")
plt.ylabel("Strata")
plt.title("Zmiany wartości funkcji kosztu")
plt.legend()
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Val Accuracy")
plt.xlabel("Epoka")
plt.ylabel("Dokładność")
plt.title("Zmiany dokładności modelu")
plt.legend()
plt.grid()

plt.show()
```

✓ 11.0s



```python
import tensorflow as tf
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
x_train, x_test = x_train.reshape(-1, 28*28) / 255.0, x_test.reshape(-1, 28*28) / 255.0

model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation="relu", input_shape=(784,)),
    tf.keras.layers.Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])

history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test), verbose=0)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Val Loss")
plt.xlabel("Epoka")
plt.ylabel("Strata")
plt.title("Zmiany wartości funkcji kosztu")
plt.legend()
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Val Accuracy")
plt.xlabel("Epoka")
plt.ylabel("Dokładność")
plt.title("Zmiany dokładności modelu")
plt.legend()
plt.grid()

plt.show()
```



3. Wnioski

Porównanie GD, SGD, Momentum, RMSProp i Adam pokazuje, że adaptacyjne metody (zwłaszcza Adam) uzyskują na ogół najszybszą i najstabilniejszą konwergencję przy szerokim zakresie współczynników uczenia, co czyni je praktycznym wyborem w większości zadań. Momentum skraca drogę do optimum na gładkich funkcjach, lecz wymaga ostrożnego doboru η, natomiast czysty SGD, choć prosty, silnie reaguje na szum gradientu i wymaga mniejszych kroków. Eksperymenty z różnymi funkcjami celu potwierdzają, że metody adaptacyjne lepiej radzą sobie z dolinami o różnych skalach nachylenia, a wizualizacje w TensorBoard ujawniają stabilniejsze rozkłady wag i gradientów w przypadku Adam i RMSProp. W praktyce dobrym punktem wyjścia jest Adam z domyślnymi parametrami, który zwykle dorównuje lub przewyższa inne algorytmy; alternatywy warto rozważyć głównie w scenariuszach, gdzie kontrola nad szybkością uczenia lub regularność ruchu w przestrzeni parametrów jest krytyczna.